VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

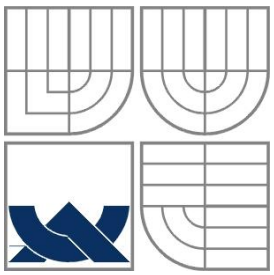
ZAROVNÁVÁNÍ ČÁSTÍ DNA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

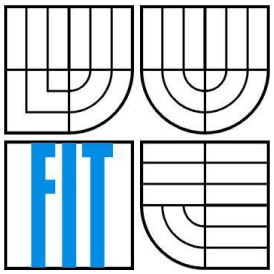
AUTOR PRÁCE
AUTHOR

Václav Pejř

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ZAROVNÁVÁNÍ ČÁSTÍ DNA

ALIGNMENT OF DNA PARTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Václav Pejř

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jaroslav Rozman

BRNO 2010

Abstrakt

Tato práce si klade za cíl zjistit, jaké jsou možnosti v oblasti zarovnávání DNA sekvencí. Na základě těchto zjištění nalézt nejlepší řešení s ohledem na rychlost výpočtu a kvalitu zarovnání. Následně toto řešení implementovat a vytvořit tak fungující program, který bude zarovnání provádět. Práce se nejprve zaměřuje na uvedení do problematiky týkající se biologie, DNA a genetiky. Po uvedení následuje přehled algoritmů, které se pro zarovnávání používají, jejich zhodnocení a výběr nejvhodnějšího algoritmu pro implementaci. Dále se také práce zaměřuje na oblast využití paralelního programování pomocí knihoven OpenCL. Zarovnání se provádí nad mnoha sekvencemi současně, zkoumají se tedy metody jak toto zarovnání provádět a jak dosáhnout nejlepších výsledků.

Abstract

This thesis deals with finding the possibilities within the sphere of alignment of DNA sequences. Based on these findings, the best solution should be found with regard to the quickness of computation and quality of alignment. Following this I intend to implement and thus create a functioning program that will do the alignment. The thesis starts with introducing issues dealing with biology, DNA and genetics. The introduction is followed by a survey of algorithms that are used for alignment, their evaluation and selection of the most appropriate algorithm for the implementation. The thesis also focuses on the usage of parallel programming by means of OpenCL libraries. The alignment is being done above many sequences at the same time, so that the methods how this process can be done and how to reach the best results are being examined.

Klíčová slova

DNA, báze, adenosin, cytosin, guanin, thymin, sekvence, biologie, genetika, Smith-Watermann, Needleman-Wunsch, blast, clustal, CUDA, OpenCL, kernel, paralelismus, zarovnání.

Keywords

DNA, basis, adenosin, cytosin, guanin, thymin, semence, biology, genetics, Smith-Watermann, Needleman-Wunsch, blast, clustal, CUDA, OpenCL, kernel, paralelism, alignment.

Citace

Pejř Václav: Zarovnávání částí DNA, bakalářská práce, Brno, FIT VUT v Brně, 2010

Zarovnávání částí DNA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením ing. Jaroslava Rozmana.
Další informace mi poskytli členové univerzity Des Higgins z Dublinu v Irsku.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Pejř
Datum (18. května 2010)

Poděkování

Tato práce by nevznikla nebýt Ing. Jaroslava Rozmana, tímto bych mu chtěl poděkovat za jeho vstřícný přístup a podporu při tvorbě této práce.

Obsah

1 Úvod

V názvu práce je obsažena zkratka DNA (deoxyribonukleová kyselina). Tato zkratka se týká jak nás, tak všeho živého na planetě Zemi. Pro všechny organismy je něco jako identifikační kód, podle kterého se jednotlivé druhy i samotní jedinci odlišují. DNA se v posledních letech dostává stále více ke slovu především v oblasti zdravotnictví, kde vznikají nové léky a léčebné postupy. Do budoucna se plánuje dokonce nahrazování nebo léčba lidských orgánů svými vlastními, které budou růst mimo tělo pacienta. Dále jsou velmi významné genetické mutace u rostlin (záměrně vyvolané člověkem), které přinášejí větší odolnost proti škůdcům a vyšší výnosy. Mohou tak pomoci při řešení otázky nedostatku potravin ve světě. Jak je tedy vidět, význam DNA nabývá rok od roku na síle a práce s DNA se staví na vrchol mnoha vědních disciplín.

Jelikož dekódovaný řetězec DNA má povětšinou několik miliard znaků, je nepředstavitelné ruční zpracování. S využitím počítačů dostává práce s DNA nový rozměr. Je možné během pár chvil data zpracovat, porovnat, nebo v nich cokoliv vyhledat. A právě porovnávání více částí DNA kódu je hlavní náplní této práce. Cílem je pak vytvořit program, který dokáže vůči sobě zarovnat několik různých částí DNA.

V první kapitole nazvané „Základní pojmy“ jsou vysvětleny pojmy nutné pro pochopení principů, kterých tato práce využívá. Jedná se o výrazy jako je DNA, kde je vysvětleno z čeho se skládá a kde je obsažena. Dále je vysvětlen pojem genetika a genetika člověka, kde se čtenář dozví základní informace týkající se především dědičnosti. V poslední podkapitole je uvedeno co to vůbec zarovnávání je a jak se přibližně provádí.

Druhá kapitola nazvaná „Zarovnávání“ se zaměřuje na algoritmy, které se pro tuto techniku používají. Nejprve jsou vysvětleny dva základní, které se staly jakoby základním kamenem pro stávající moderní algoritmy. Po tomto vysvětlení práce pokračuje přehledem algoritmů moderních, které se v současnosti řadí na vrchol nejčastěji používaných algoritmů pro práci s DNA.

Třetí kapitola se soustředí na uvedení přehledu formátů, ve kterých je možné DNA počítačově přenášet a zpracovávat. Jedná se o nejčastěji používané formáty, které se vyskytují i v genetických databázích. A na veřejně dostupných zdrojích genetických dat.

Čtvrtá kapitola nazvaná „OpenCL“ se zaměřuje na nejmodernější metody paralelního programování. Je zde popsána technologie, která se používá pro náročné aplikace, u kterých je možné jednotlivé výpočty provádět paralelně. V kapitole získá čtenář přehled o tom, na jakých principech *OpenCL* pracuje, kam ukládá data a jak lze tuto technologii použít v praxi.

Předposlední kapitola se již zabývá návrhem a implementací programu na zarovnávání DNA sekvencí. Čtenář se zde dozví, jaký algoritmus se stal nejvhodnějším kandidátem a proč. Jaká technika byla při implementaci použita. A také se dočte, jakých výsledků bylo dosaženo v porovnání

s existujícími programy na zkušebních datech. Získá tedy přehled o veškeré implementaci zvoleného řešení a také o tom, nakolik byla implementace úspěšná.

V závěrečné kapitole je uvedeno celkové shrnutí práce, kde hraje důležitou roli porovnání výsledků s předpoklady. Dále se čtenář dozví, v čem implementace selhala a v čem naopak překvapila, k čemu vůbec celá práce dospěla a zda má nějaký význam do budoucna. Také je zmíněno, jak lze na práci navázat a dokázat dosažené výsledky využít a vylepšit.

2 Základní pojmy

Jelikož se tato práce zabývá technikami pro provádění zarovnání DNA kódu, je nutné nejprve pochopit, co vše se vůbec za DNA kódem a potřebami jeho zarovnání skrývá. Popíšeme si tedy nejdůležitější vědní obory, které za touto problematikou stojí. Obecně nás zajímají obory biologické, které se týkají především genetiky a biologie obecně.

2.1 Biologie

Kořeny vědního oboru biologie sahají až hluboko do starověku. Jednalo se o zemědělství a lékařství, které se v této době začalo zvolna rozvíjet. Na překotnější rozvoj museli obory čekat poměrně dlouhou dobu, neboť nastala jistá stagnace hlavně z důvodu nedostatečného technického pokroku. Až v novověku, s příchodem renesance došlo k významnému urychlení vývoje vědy mimo jiné díky některým vynálezům. Jednalo se především o vynalezení mikroskopu někdy během 17. století. Tento objev již předznamenal vznik biologického oboru. Díky mikroskopu došlo k objevení rostlinné i živočišné buňky, objevu některých mikroorganismů, především bakterií, a byly položeny základy anatomie.

Významným krokem vpřed byla evoluční teorie Charlese Darwina (1809 – 1882), která zodpověděla otázku vývoje druhů za pomoci přírodního výběru. A s další teorií přišel Johann Gregor Mendel (1822 - 1884), který roku 1866 publikoval své výsledky zkoumání křížení rostlinných druhů a díky těmto výsledkům definoval základní principy přenosu genetické informace. Tímto se stal zakladatelem moderní genetiky.

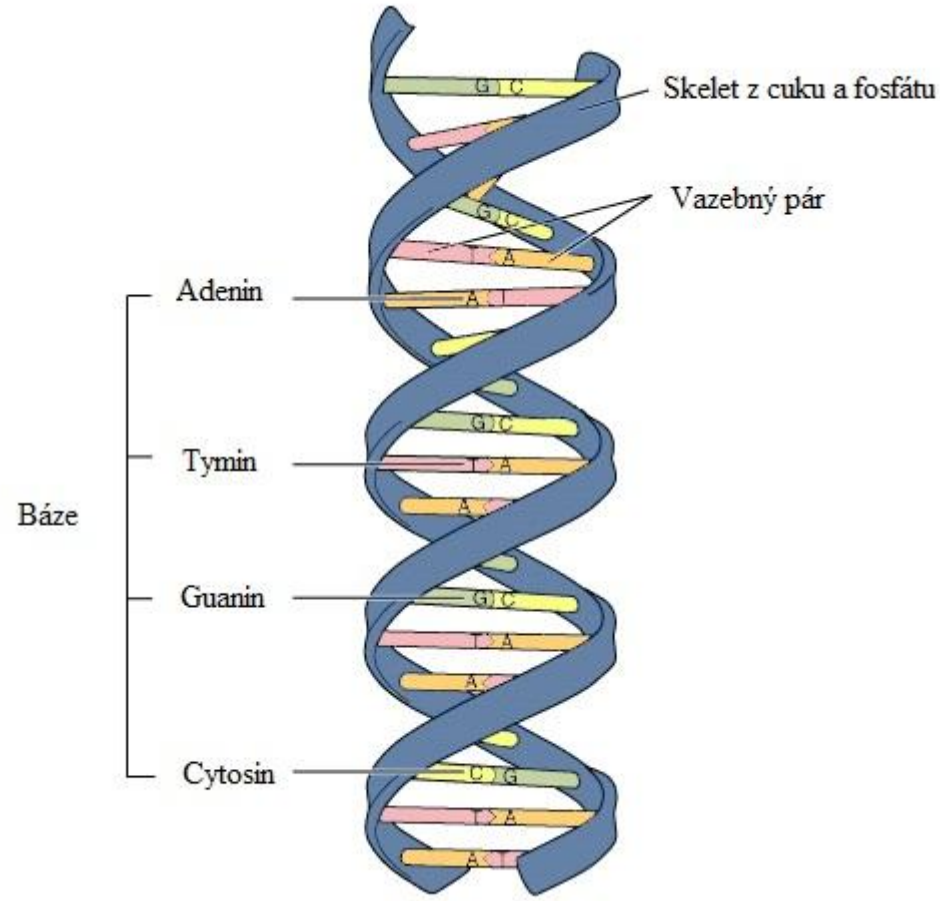
K doposud největšímu rozmachu biologie dochází od počátku 20. století a to díky novým experimentálním postupům a přístrojům. Poprvé se člověk ve svém bádání v oblasti buněčných organismů dostává až na molekulární úroveň. Roku 1961 dochází k definování proteosyntézy (DNA-RNA-bílkovina). Od 70. let je možná přímá analýza DNA, která způsobila převrat ve zkoumání života. Tato analýza vedla k objevení nových proteinů a genů, jejichž evoluci bylo možné mimo jiné zkoumat i odděleně mimo živé buňky. Umožnila také rychlý vývoj molekulární biologie, která se zabývá vztahem struktury nukleových kyselin a proteinů k funkcím a vlastnostem buňky. Navazujícím oborem se stala genomika, jejímž cílem je stanovit úplnou dědičnou informaci organismů. Poznatky genomiky jsou důležitým zdrojem informací pro srovnávací biologii.

2.1.1 DNA

Až na výjimku nebuněčných organismů je DNA součástí všeho živého. Z technického pohledu je v ní zakódován program života organismu. Najdeme zde plány pro syntézu nukleových kyselin a bílkovin, které ovlivňují vlastnosti každé buňky v těle. A stejně tak v každé buňce nalezneme program pro všechny buňky v těle. Tedy každá buňka nese kompletní informaci o celém organismu, ale pro svoji vlastní potřebu používá jen velmi malou část tohoto kódu. V rámci jednoho organismu nalezneme tedy vždy totožný DNA kód. V porovnání s jinými živými organismy je však tento DNA kód unikátní. Zřejmě jedinou výjimku v tomto pravidlu tvoří ty organismy, které se rozmnožují klonováním. Těchto organismů není mnoho a jejich společenstva naráží často na problém, který zapříčiňuje právě shoda DNA kódu. Tyto organismy pokud jsou napadeny například škodlivým virem, proti kterému nemají obranu, má to fatální následky pro celé společenstvo. I z tohoto důvodu na planetě převládá skupina organismů, kde se kříží DNA od dvou jedinců, čímž získává variabilitu celé společenstvo.

DNA se dělí na menší části, tzv. geny. Gen je schopen při dělení buňky vytvářet své vlastní kopie, které se přenáší do dalších generací. Dále dokáže kódovat jednu nebo i více bílkovin současně. Vyskytuje se v několika formách, tzv. alelách. Alely jsou buď dominantní, nebo recesivní. Podle nich se následně řídí to, jak se gen projeví. U prokaryotních buněk se nachází DNA v plazmidech a na tzv. prokaryotickém chromozómu. Mezi prokaryotní organismy patří především ty jednodušší, jako jsou různé viry a bakterie. U eukaryotních je DNA povětšinou uvnitř jádra na chromozómech a částečně i v mitochondriích (živočišné buňky) nebo v chloroplastech (rostlinné buňky).

Pokud půjdeme ještě hlouběji do struktury DNA, dojdeme až ke stavebním kamenům tohoto celku, kterými jsou báze. Báze se dělí na purinové a pyrimidinové. Purinové jsou adenosin (A) a guanin (G). Pyrimidinové pak cytosin (C) a thymin (T). Báze se vzájemně spojují a tvoří tzv. vazebné páry, které spojují oba řetězce molekuly DNA do pravotočivé šroubovice pomocí vodíkových můstků. Takto spojené řetězce jsou mnohem odolnější a stabilnější než pouze řetězec jeden. Báze se vzájemně slučují podle pravidel komplementarity, kdy se spojuje adenosin a tymin, druhou dvojici pak tvoří cytosin a guanin. Díky této vlastnosti lze snadno kopírovat DNA jako celek. Dojde k oddělení vazeb. Vzniknou dva samostatné řetězce, na které se podle pravidel komplementarity naváží jednotlivé báze a takto vzniká nová kopie, která má již opět tvar pravotočivé šroubovice tvořené dvěma řetězci spojenými pomocí vodíkových můstků.



Obrázek 1: DNA dvoušroubovice

2.1.2 Genetika

Genetika je vědní obor zabývající se dědičností, geny a proměnlivostí organismů. Základy tomuto vědnímu oboru položil Johann Gregor Mendel.

Jelikož se zabývá dědičností všech organismů jako celku, dělí se na kategorie se zaměřením na konkrétní oblast zkoumání. Genetika se uplatňuje především při výzkumu rakovinného bujení, imunitního systému a imunitních reakcí a v mikrobiologickém výzkumu.

Genetická informace určuje budoucí anatomickou stavbu organismu, určuje, jaké látky budou účastníky biochemických a fyziologických procesů v organismu a v neposlední řadě je nepostradatelnou součástí pohlavního i nepohlavního rozmnožování. S využitím genetické informace vyvstávají nové možnosti. V praxi se tedy můžeme setkat s uplatněním genetiky při odhalování pachatelů trestných činů, nebo při potvrzování otcovství, případně i při identifikaci tělesných ostatků.

Obor genetika je velmi rozsáhlý a značně se liší jednotlivé oblasti výzkumu v tomto oboru. Proto byl rozdělen na mnoho podoborů, jednak jsou rozděleny podle metod výzkumu, kdy se jedná například o obory, jako jsou: toxikogenetika, nutrigenetika, farmakogenetika, populační genetika a

evoluční genetika. Dále kromě těchto metod rozlišujeme genetiku ještě podle objektu výzkumu. Zde se jedná například o gentiku virů, prokaryot, živočichů, rostlin a především o genetiku člověka.

Nás zajímá v této práci genetika jak rostlin, tak i živočichů a okrajově i virů a dalších prokaryot. Zaměřme se tedy především na genetiku člověka. Ta má z těchto oborů nejvíce specifických vlastností i omezení, která plynou z morálních zásad společnosti.

2.1.3 Genetika člověka

Jelikož je genetika především disciplína založená na experimentech, pak jasně vidíme omezení, kterým člověk jako objekt výzkumu je. Jedná se převážně o etické zásady, kterými se společnost řídí a není tedy přípustné dělat na člověku různé genetické experimenty. Z toho důvodu genetika člověka jako experimenty může brát pouze události, které se odehrávají spontánně. Tedy například sňatky a s nimi spojení potomci, jako ukázka genetického křížení. Dále například některé havárie s nebezpečnými látkami, které mohou způsobit genetické mutace. Zde se jedná například o jadernou katastrofu v Černobylu. Další nevýhodou jsou velké věkové generační rozdíly. Například při porovnávání rozdílu 2 generací (děda a vnuk) se jedná o rozdíl přibližně 40let, zatímco u myší se jedná asi o 2 měsíce a u mikroorganismů asi jen o 2 hodiny. Toto ale nejsou stále všechny problémy, se kterými se musí genetici potýkat. Jedná se také o malý počet potomstva, kdy v dnešní Evropě je průměrná porodnost pod hranicí 2 dětí na jednu rodinu. Kdežto například u myší se jedná o 4 až 11 mláďat a to až 8krát do roka. Což je velmi propastný rozdíl v množství objektů, které lze zkoumat. A jako poslední nevýhodu bych uvedl nemožnost dodržení stejným vnějších podmínek, tedy nelze posuzovat jednotlivé samovolné experimenty člověka stejně jako ty, které jsou prováděné v laboratorních podmínkách, kde si jsou genetici jistí, že na experiment nebude mít okolí žádný vliv.

Kromě negativních věcí, které ztěžují genetický výzkum, jsou i věci, které výzkum ulehčují. Jedná se například o velkou genetickou variabilitu člověka. A také o to, že jsou vedeny nejrůznější statistiky několik desítek let, mnohdy až několik staletí. Zde se jedná o nejrůznější zdravotnické nebo demografické záznamy. A díky počítačovému zpracování údajů mohou genetici porovnávat nejrůznější údaje z celého světa.

Základním cílem genetiky člověka je zkoumání jeho genetické variability, kterou dělíme na patologickou a nepatologickou. Patologickou variabilitou se zabývá lékařská genetika. Která zkoumá geneticky podmíněné choroby člověka. Jaký vliv na tyto choroby má dědičnost a jak lze těmto chorobám nejlépe předcházet.

2.2 Zarovnávání

V této části se snažím čtenáři vysvětlit vše o zarovnávání od základů až po konkrétní a moderní algoritmy. Pro potřeby zarovnávání bylo vyvinuto mnoho algoritmů pracujících na odlišných principech. Rád bych popsal pár zásadních algoritmů, které se týkají oblasti této práce a současně byly použity při modernizaci a vývoji těch nejlepších algoritmů současnosti.

2.2.1 Základní principy

Již několik desítek let dokážeme přečíst genetickou informaci téměř jakéhokoliv živého organismu. Pouhé přečtení jednotlivých znaků nám však neřekne vše, co bychom potřebovali znát. Dozvíme se z kódu, z jakých genů se DNA skládá a jaký vliv mají jednotlivé geny na celý organismus. Ale v případě, že chceme zjistit, jak se liší například DNA dvou odlišných živočišných druhů, nebo více jedinců stejného druhu, pak je jednou z možností zkoumat ručně miliardy znaků. Druhou a mnohem rychlejší je použití specializovaného programu, který toto porovnání udělá za nás. Tedy zarovnání DNA sekvencí je důležité pokud zkoumáme vlastnosti, ve kterých se organismy vzájemně shodují či odlišují.

Zarovnávání se tedy používá v případě porovnávání dvou a více sekvencí, případně i při vyhledávání částí sekvencí. Pokud bychom měli jen jednu osamocenou sekvenci a chtěli ji zarovnat, vždy musíme mít sekvenci další, vůči které zarovnání provedeme. Princip je jednoduchý. Musíme jen rozhodnout, co udělat v místech, kde se sekvence plně neshodují. Jako příklad si zvolme sekvence z obrázku 2. Vidíme, že se neshodují na pátém znaku, kdy stojí proti sobě *Guanin* s *Cytosinem*, tudíž nemohou vytvořit společnou vazbu a je jasné, že musíme jejich postavení vůči sobě změnit, abychom mohli sekvence vůči sobě porovnat.

```
A A C T G A A
A A C T C A A
```

Obrázek 2: Dvě sekvence

```
A A C T G – A A
| | | |     | |
A A C T -- C A A
```

Obrázek 3: Dvě zarovnané sekvence

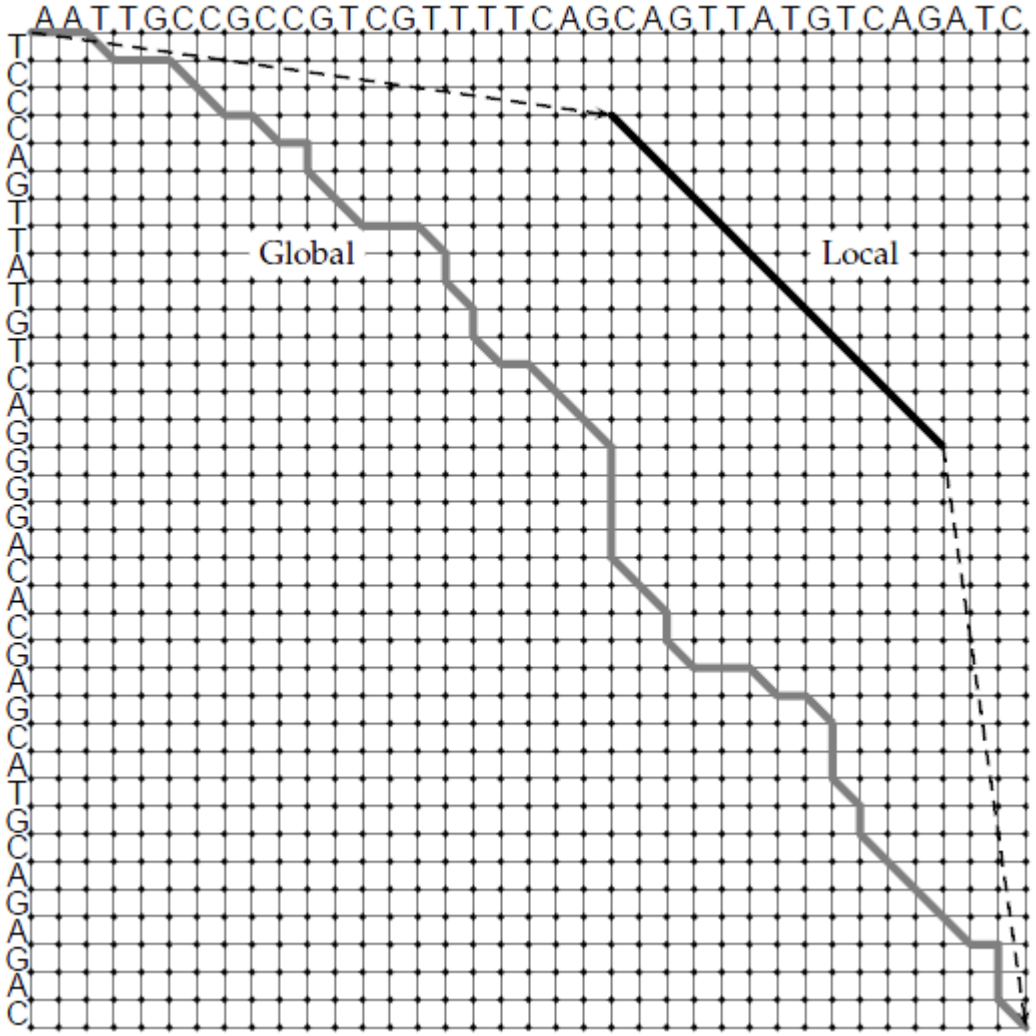
Na výše uvedeném příkladu se zdá, že zarovnávání je v celku jednoduchá věc. V tomto případě tomu tak je, ale pokud místo dvou sekvencí jich budeme vůči sobě zarovnávat více, složitost rozhodování poroste. Abychom dokázali jednoznačně určit, jak danou sekvenci zarovnat, potřebujeme na to speciální algoritmus. Ten nám pomůže situaci vyhodnotit, určí nejvhodnější zarovnání tak, aby co nejvíce odpovídalo realitě. Tím se myslí, že daný kus DNA kódu, který se pokoušíme zarovnat je skutečný. Proto vzorky, které vzájemně přeskupujeme, musíme dostat do takové formy, jaká je v přírodě běžná. Nejdůležitější věcí je splnění zákonů komplementarity. Tedy

splnění vazeb, které se v DNA řetězci mohou vytvářet. Jedná se o vzájemné vazby thyminu s adeninem a guaninu s cytosinem, kdy tyto páry jsou schopny se vzájemně vázat. Nemůže dojít k samovolnému vytvoření vazby mezi jinými bázemi (ani v RNA, kde je thymin nahrazen uracilem).

2.2.2 Lokální a globální zarovnávání

Při rozhodování jaký algoritmus je nejvhodnější pro daný problém musíme znát účel jeho použití. Algoritmy se liší v několika významných věcech. Jedná se buď o algoritmy určené pro zarovnání právě dvou sekvencí, nebo o algoritmy zabývající se zarovnáním mnoha sekvencí. Dále záleží, zda provádíme zarovnání celých sekvencí, nebo zda jen vyhledáváme menší část, kterou je třeba mnohokrát zarovnat. Při vyhledávání není třeba posuzovat sekvence jako celky, proto je zde využito tzv. lokální zarovnání. A naopak je tomu v případě zarovnání celých sekvencí, pak je použito tzv. globální zarovnání. Rozdíl mezi globálním a lokálním zarovnáním můžete vidět na obrázku 4. Na obrázku je vidět, že došlo k dvěma různým zarovnáním dvou sekvencí. Jejich výsledek, který vyplývá z grafu, můžete vidět na obrázku 5 a 6. Algoritmy, které by nás k takovémuto výsledku dovedly, uvádím v kapitolách 2.2 a 2.3.

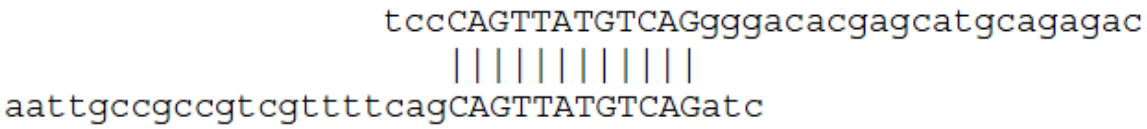
Na obrázcích je vidět, jak se může diametrálně lišit zarovnání stejných párů. Podle postavení zarovnaných znaků lze vyčíst, jak jednotlivé algoritmy hodnotí jejich vzájemnou pozici. Pokud se podíváme na obrázek 5, pak vidíme obě sekvence rozložené po celé své délce. Obě jsou zarovnány podle maximální shodnosti znaků na pozicích i poměrně daleko od sebe. Navíc je vidět snaha o to, aby byly ukončeny i započaty znaky, nikoliv mezerami. Neboť jak bylo výše zmíněno, pokud má jít o skutečné sekvence, pak musí být nejlépe zarovnány v celé své délce. Naopak je tomu v případě obrázku 6. Zde byl nalezen pouze nejdelší shodný vzorek. Z hlediska podobnosti zkoumaného páru můžeme díky tomu zjistit, že se jedna poměrně významná část DNA kódu shoduje v obou sekvencích. Tato metoda je velice vhodná, pokud je účelem vyhledávání takovýchto společných částí. Nebo naopak, pokud například globální zarovnání ukáže, že se pár shoduje minimálně, tedy, že si vzorky nejsou podobné. Pak může lokální zarovnání ukázat, že se vzorky shodují jen v jistém úseku, který je pro ně společný a zbytek kódu se liší. Oba způsoby tedy pracují na odlišných principech a požadujeme od nich odlišné výsledky. V mnoha situacích se oba tyto způsoby zarovnávání kombinují, aby se zohlednilo jak nalezení shodných částí, tak i současně globální srovnání celého páru.



Obrázek 4: Rozdíl mezi lokálním a globálním zarovnáním. Zdroj[1]

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |   || |   ||   | | | |||     || |   | |   | ||||    |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
```

Obrázek 5: globálně zarovnané sekvence. Zdroj[1]



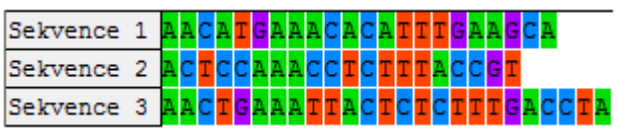
Obrázek 6: lokálně zarovnané sekvence. Zdroj[1]

2.2.3 Porovnávání párové a mnohonásobné

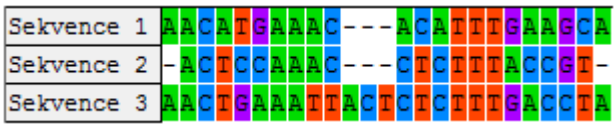
Vysvětlen byl rozdíl mezi globálním a lokálním zarovnáním. Nyní si vysvětlíme rozdílný způsob práce se sekvencemi pro případy, kdy zarovnáváme právě dvě, nebo zarovnáváme více než dvě vůči sobě.

V kapitole 2.1.1 na obrázcích 5 a 6 lze vidět typický příklad zarovnání právě dvou sekvencí. Jedná se o tzv. párové zarovnání. Kdy máme jeden pár sekvencí, kde na základě vybraného algoritmu jeho aplikací dojdeme ke konečnému řešení. Existuje ale mnoho případů, kdy potřebujeme nalézt podobnost mezi více sekvencemi současně, nikoliv párově. Důvod je jednoduchý. U párového zarovnání jsou vidět podobnosti pouze mezi dvěmi sekvencemi. Pokud potřebujeme například zjistit u pěti sekvencí, jak moc jsou rozdílné, pak párové porovnání nebude vždy dostatečné. Mohli bychom pouze porovnat každé dvě a následně kombinovat takto vzniklé, nově zarovnané s ostatními. Práce by to byla náročná na čas a výsledek by nebyl jistý. Proto byly vytvořeny specializované algoritmy, které se zabývají pouze tímto mnohonásobným zarovnáváním.

Mnohonásobné zarovnávání je založeno principielně na párovém. Jen s tím rozdílem, že párové zarovnání se použije v základní fázi hodnocení sekvencí. V dalších fázích je hodnocení kombinováno a jsou porovnávány nejrůznější kombinace znaků. Výsledkem práce algoritmu je pak vůči sobě zarovnaná celá skupina sekvencí. Pro ukázku bychom mohli uvést příklad výstupu takového algoritmu na libovolném vzorku dat. Mějme proto například tři sekvence, které jsem libovolně vytvořil. Jsou na obrázku 7, obrázek je vyňat z programu *MEGA 4.1*, ve kterém jsem následně provedl mnohočetné zarovnání pomocí algoritmu *Clustal* (bude zmíněn až v dalších kapitolách) a jeho výstup lze vidět na obrázku 8.



Obázek 7: Náhodně zvolené 3 sekvence.



Obrázek 8: Sekvence po zarovnání algoritmem Clustal v programu MEGA 4.1

Na výstupu si můžeme povšimnout, jaké části sekvencí algoritmus hodnotil kladněji a ve výsledku poukazuje na prvky společné pro všechny 3 sekvence. Podle předchozí kapitoly víme, že se zde jedná o globální zarovnání, proto některé neshodné části jsou i přesto spolu. Dalším důvodem proč algoritmus ponechal například na 4. znaku proti sobě stát adenin s thyminem je to, že znaky netvoří celou aminokyselinu. To je způsobeno především mým výběrem znaků, což nám pro ukázku nijak nevadí. Ale jistou představu o tom, jak takové mnohočetné zarovnání vypadá, jsme získali, což bylo účelem.

3 Zarovnávací algoritmy

Informace o zarovnávání získané z kapitoly 2.3 dále rozvedeme do konkrétní podoby algoritmů. Čtenář se dozví, na jakém principu fungují základní algoritmy a následně v jakém směru došlo postupem času k jejich vylepšení a k jakým změnám.

3.1 Algoritmy základní

Abychom získali přehled o tom, z čeho dnešní algoritmy vycházejí, na čem se zakládají, uvedu zde dva nejdůležitější algoritmy v této oblasti vůbec. Čtenář se zde dozví, jak fungují a jak se dají aplikovat. Získá o nich i stručný historický přehled a přehled z oblasti jejich použití.

3.1.1 Needleman-Wunschův algoritmus

Tento algoritmus byl poprvé publikován v roce 1969 Saul B. Needlemanem a Christianem D. Wunchem. Je to první zástupce dynamického programování na poli bioinformatiky. Zabývá se globálním zarovnáváním dvou sekvencí. K zarovnávání je použita hodnotící matice, která určuje hodnocení postavení prvků v obou sekvencích. Pro vytvoření hodnotící matice lze použít dvourozměrné pole, kde definujeme hodnocení párů, které se mohou v sekvencích vzájemně potkat.
Například pro zarovnávání DNA tato matice bude mít 4sloupce a 4řádky označené jednotlivými znaky, které zastupují aminokyseliny (obrázek 9). Při následném zarovnávání jsou jednotlivé páry ohodnoceny podle této matice. Zkoumáme co nejvíce možných variant a porovnáváme výsledky hodnocení. Platí zde, že čím vyšší číslo dostaneme, tím je zarovnání lepší. Při porovnávání je použita metoda tzv. *gap penalty*, kdy za vloženou mezeru jsme penalizováni. Nejčastěji je tato hodnota rovna nejhůře hodnocenému páru.

-	A	C	T	G
A	10	-1	-3	-4
C	-1	7	-5	-3
T	-3	-5	9	0
G	-4	-3	0	8

Obrázek 9: hodnotící matice

Pro pochopení jak algoritmus hodnotí, uvedu příklad, který pracuje s hodnotící maticí z obrázku 9 a používá *gap penatly* na hodnotě -5 (*gap penalty* budeme značit malým písmenem *d*):

Mějme Dvě sekvence A a B, kde A: ACTCCTAGG a sekvence B: ATTCG. Tyto dvě máme co nejlépe vůči sobě zarovnat. Jedná se o globální zarovnání, tedy měli bychom se držet zásady

upřednostnit takové hodnocení, které začíná a končí aminokyselinou, nikoliv mezerou. Pokud by došlo ke shodě, pak se budeme řídit tímto pravidlem.

Průběh:

1 - vybereme nejpravděpodobnější varianty ke zkoumání

2 - ohodnotíme zkoumané varianty

3 - vybereme nejlepší variantu

1 – vybrané varianty: sekvence A je neměnná, zůstává v původním stavu a vůči ní zarovnáváme sekvenci B, pro kterou volíme vhodné varianty:

1. varianta: A - T - - T C - G
2. varianta: A - T - - T - C G

2 – hodnocení provedeme pro každý znak zkoumané varianty:

1. varianta: S(A,A) + d + S(T,T) + d + d + S(T,T) + S(C,A) + d + S(G,G) =
   = 10 – 5 + 9 – 5 – 5 + 9 – 1 – 5 + 8 = 15
2. varianta: S(A,A) + d + S(T,T) + d + d + S(T,T) + d + S(C,G) + S(G,G) =
   = 10 – 5 + 9 – 5 – 5 + 9 – 5 – 3 + 8 = 13

3 – Dle hodnocení vyplynulo, že vyššího hodnocení dosáhla varianta 1. Proto by zde byla vybrána jako nejlepší možnost pro zarovnání. Výsledkem bychom tedy získali zarovnání jaké je na obrázku 10.

Obrázek 10: zarovnané sekvence

3.1.2 Smith-Watermanův algoritmus

Pochází z roku 1981 a je založen na principu předešlého, tzv. Needleman-Wunschova algoritmu. Zabývá se spíše lokálním než globálním zarovnáním. Pro zarovnávání používá opět pouze dvě sekvence. Pokud bychom chtěli porovnat sekvencí více, je nutné vycházet pouze z porovnání dvojic. Což není optimální a nehledá se žádná spojitost ve více sekvencích vůči sobě. Proto by mohli být výsledky u porovnání více sekvencí zavádějící a nepřesné. Přesto je Smith-Watermanův významným algoritmem, který posloužil k dalšímu vylepšování a rozvoji mnoha dalších.

Jeho princip spočívá opět ve vytvoření hodnotící matice. Ta určuje postavení jednotlivých znaků vůči sobě. Po ohodnocení všech prvků procházíme matici od nejvyššího indexu po nejnižší a

cestu volíme dle ohodnocení. Po dokončení průchodu máme optimálně zarovnané sekvence (pouze lokálně). Jelikož se jedná o lokální zarovnání, je algoritmus používán nejčastěji při vyhledávání podsekvencí, kde je tento způsob zarovnání nejvíce potřebný.

Matice je tvořena následovně:

$H(i,0) = 0,\ 0 \leq i \leq m$

$H(0,j) = 0,\ 0 \leq j \leq n$

$$H(i,j) = \begin{cases} 0 \\ H(i-1,j-1) + w(ai,,bj) \text{ shoda/neshoda} \\ \max H(i-1,j) + w(ai, -) \quad \text{vymazání} \\ H(i,j-1) + w(-, bj) \quad \text{vložení} \end{cases}, 1 \leq i \leq m,\ 1 \leq j \leq n$$

Kde: *a, b* řetězce nad abecedou ∑ {A,C,T,G} (konkrétně pro DNA, jinde se může lišit)

m počet znaků řetězce *a*

n počet znaků řetězce *b*

H(i,j) maximální hodnocení podobnosti mezi znaky *a[1..m]* a *b[1..n]*

w(c,d), c,d $\in \sum \cup \{'-'\}$, kde *'–'* zastupuje nahrazení znaku mezerou

Příklad: mějme dvě nezarovnané sekvence, u nichž chceme získat nejlepší zarovnání.

Sekvence1: ACACACTA

Sekvence2: AGCACACA

Vytvoříme si matici o rozměrech velikosti počtu znaků našich sekvencí, tedy konkrétně o rozměrech 8x8 plus ještě jeden rozměr pro inicializaci, tedy prakticky matice 9x9. V inicializačním kroku nastavíme hodnoty prvního sloupce i prvního řádku na nulu. Následně zkoumáme nastavení prvního neohodnoceného místa, tedy místa na souřadnici [1,1]. Porovnáme znaky na těchto souřadnicích. Oba znaky jsou shodné, došlo ke shodě. Dle pravidel dostáváme výsledek jako součet hodnoty na souřadnici [i-1, j-1] (tedy [0,0]) a hodnoty vzniklé porovnáním. Proto součet bude 0 + 2 = 2. Tuto hodnotu zapíšeme na zkoumanou souřadnici. Takto pokračujeme pro celou matici, kdy v případě neshody postupujeme dle zadaných pravidel. Tedy vybíráme nejvyšší hodnotu, která vznikne po aplikování pravidel. První pravidlo nám určuje, že nejnižší hodnotou matice je 0, tedy nelze se v hodnocení dostat do záporných čísel. Toto omezení lze upravit podle potřeb aplikace algoritmu.

Pro úplnost uvádím výsledek aplikování pravidel a takto vzniklou matici, kterou lze nalézt taká na adrese viz. zdroj [2].

H =

	---	A	C	A	C	A	C	T	A
---	0	0	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1	0	2
G	0	1	1	1	1	1	1	0	1
C	0	0	3	2	3	2	3	2	1
A	0	2	2	5	4	5	4	3	4
C	0	1	4	4	7	6	7	6	5
A	0	2	3	6	6	9	8	7	8
C	0	1	4	5	8	8	11	10	9
A	0	2	3	6	7	10	10	10	12

3.2 Algoritmy vylepšené

Dva výše uvedené algoritmy jsou alfou a omegou veškerého zpracování DNA sekvencí. Nejedná se zde pouze o algoritmy zarovnávací, ale také o vyhledávací. Tyto nové, modernější algoritmy využívají principů výše uvedených jako základ své vlastní práce a bez jejich znalosti bychom se tedy neobešli. Pro ukázku dále uvedu jeden vyhledávací a jeden zarovnávací algoritmus. V současné době se jedná o algoritmy, které jsou na čele, co se týče rychlosti a přesnosti své práce.

3.2.1 Blast

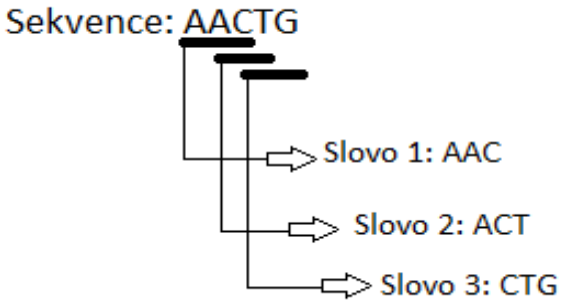
Tento algoritmus jsem vybral jako zástupce dnešních nejlepších vyhledávacích algoritmů. Je nástupcem mnoha předešlých, ze kterých si vzal vždy to podstatné a k tomu něco nového navíc. Zakládá se především na principech algoritmu *fasta*, tyto principy budou vesměs vysvětleny spolu s vlastnostmi tohoto algoritmu.

Název algoritmu je zkrácený popis jeho funkce, tedy základní lokální zarovnávací vyhledávací nástroj (*basic local alignment search tool*). Jeho funkci lze kombinovat v několika způsobech získávání informací. Jedná se především o vyhledávání podobností různých částí DNA kódů. Jednou možností je, kdy máme několik proteinů, tvořených několik desítkami znaků a hledáme v DNA řetězci nějakého organismu podobnosti. Tedy snažíme se nalézt, zda se tyto proteiny vyskytují v zadaném pořadí a zda se opakují. Druhým způsobem použití je nechat proti sobě dvě celé sekvence, které porovnáme tímto algoritmem. Může se jednat například o porovnání lidské DNA s DNA myši. Výsledkem budou podobné oblasti v obou vzorcích DNA, tedy kde jsou si tyto živočišné druhy podobné. Tato vlastnost algoritmu vychází z principu jeho funkčnosti, jelikož se jedná o lokální

zarovnávání, pak se budou vyhledávat nejlépe shodné části bez ohledu na DNA jako celek. To je pro tento případ velice výhodné a často aplikované. Nyní bych rád uvedl, na jakých principech tento algoritmus vlastně funguje.

Nejprve ze zadané sekvence ořeže ty části, které by mohli zmást program. Jedná se vždy o skupinky několika osamocených znaků, které by získali díky malému počtu vysokého ohodnocení a mohli by tak ovlivnit výsledek.

Dále rozdělíme zadanou sekvenci na x-znaková slova, kdy za x dosadíme jakékoliv číslo. Například dosaďme číslo 3, pak bude vypadat rozdělení podobně jako na obrázku 11.

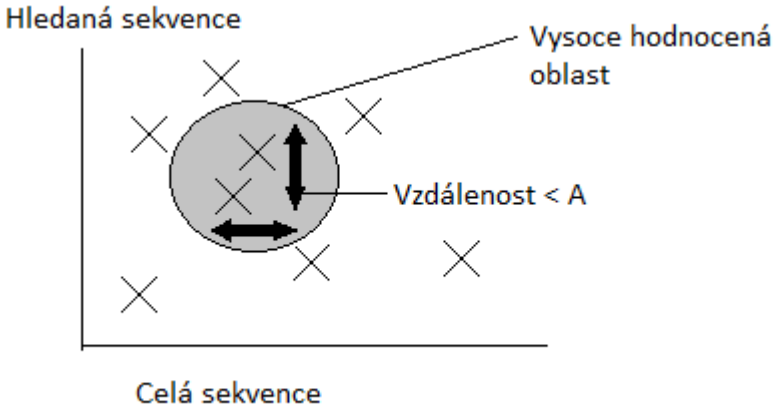


Obrázek 11: Rozdělení hledané sekvence na slova o n-znacích

Třetím krokem se *blast* odlišuje od algoritmu *fasta* značným způsobem. Jedná se zde totiž o získání nejlepších kombinací slov z kroku 2, které by mohli zaručovat kladný výsledek hledání. K tomuto vyloučení nevhodných slov dochází pomocí skórovací matice, kdy jsou porovnány hodnocení jednotlivých n-tic a ty nejhůře hodnocené nejsou při provádění samotného prohledávání použity. Což snižuje počet samotných porovnání a tím i zvyšuje rychlost programu a přitom nedochází ke snížení kvality výsledného zarovnání. Díky tomuto kroku dochází ke znatelnému urychlení vůči algoritmu *fasta*.

Po odstranění nepotřených slov dochází k organizaci zbylých, tedy vhodných slov do vyhledávacího stromu. S tímto stromem se pak pracuje při prohledávání, kdy se z databáze sekvencí, nad kterými chceme prohledávání provádět, vezme vždy jedna konkrétní a zkoumají se jednotlivá n-znaková slova. Pokud dojde k přesné shodě slova s částí prohledávané sekvence, pak se ponechá tato část nezarovnaná. Při částečné shodě dochází k zarovnání nalezené části tak, aby dosáhla nejvyššího možného hodnocení, čímž se zajistí maximální pravděpodobnost, že se opravdu jedná o hledanou část.

Další krok spočívá v rozšíření přesné shody do vysoce hodnocené párové oblasti (obrázek 12). V originále je tato oblast označovaná jako HSP (*high-scoring segment pair*).



Obrázek 12: Pozice přesné shody.

Ke konečnému výsledku se algoritmus dostane až za použití vytvořených HSP regionů. Tyto regiony jsou zapotřebí především kvůli počtu hledaných slov. HSP tedy balancuje jednak počet hledaných výrazů, který může být velmi významný, při použití krátkých slov (jak jsme uváděli například slova složená z 3 znaků). A přesnost vyhledávání na celém úseku DNA sekvence. Pokud bychom totiž jen zkoumali jednotlivé n-tice, dosáhneme obrovského množství shod. Tomuto lze předcházet zadáním delších n-tic, kdy ale přicházíme o možnost zkoumání na úrovni jednoho proteinu. Abychom nebyli o tyto možnosti ochuzeni, byl právě vymyšlen tento princip pomocí HSP, který umožňuje vyhledání v krátkém čase na délku klidně pouze jednoho proteinu.

3.2.2 Clustal

Tento algoritmus se skládá ze tří základních částí:

1 – Všechny páry sekvencí jsou zarovnány samostatně vůči sobě v takovém pořadí, aby bylo možné sestavit matici vzdáleností jednotlivých znaků (je závislá na neshodě v jednotlivých párech).

2 – Ze získané matice je vypočítán řídící strom.

3 – Sekvence jsou postupně zarovnány podle větvení řídícího stromu

Matice vzdáleností

Klasicky se v programech používajících algoritmu *Clustal* počítá matice vzdáleností pomocí rychlých aproximačních metod. Což umožňuje porovnat opravdu velké množství párů. Skóre se pak počítá jako počet shod (délka shodných znaků pro uznání shody se liší podle typu, délka 1 až 2 znaky pro proteiny a 2 až 4 pro nukleotidové sekvence) mezi dvěma sekvencemi, od kterých se odečítá za každý rozdíl mezi nimi. Poměr kladných a záporných bodů není hodnocen 1:1, ale liší se v závislosti na délce znaků ve shodě. Tedy pro jednu shodu přičteme například 5bodů a pro neshodu v jednom znaku pak odečteme 1bod.

Řídící strom

Řídící stromy jsou používané pro mnohonásobné zarovnání (zarovnání více sekvencí vůči sobě jako celek, nikoliv jako jednotlivé páry). Jsou vypočítávány na základě matice vzdáleností použitím metody sousedské podobnosti. Jednotlivé větve stromu označují ohodnocení dané sekvence povětšinou pro každý znak zvlášť. Do hodnocení již vstupuje i podobnost jednotlivých sekvencí. Neboť celkové ohodnocení dané větve závisí i na podobnosti (cestě) s ostatními větvemi. Pak se k samotnému hodnocení přičte ještě poměrná část podobné větve, která má část cesty společnou s hodnocenou větví. Zde dochází k celkovému upravení hodnocení pro všechny sledované sekvence. V této finální podobě strom zůstává až do konce běhu programu. Další zpracování z tohoto stromu už pouze čte, tedy nemění jeho strukturu ani hodnocení větví.

Postupné zarovnání

Postupné zarovnávání používá řídícího stromu jako hlavního prvku v rozhodování o výsledném zarovnání sekvencí. Kdy na základě tohoto stromu, se vybere vždy nejlépe hodnocená varianta. To znamená, že pokud máme například 10 sekvencí a sestavený řídící strom. Pak porovnáváme hodnocení u všech sekvencí prvního znaku (u proteinů i dva znaky a u nukleotidových sekvencí až 4 znaky) a následně výpočtem zjistíme, který výsledek je nejlépe hodnocen pro většinu sekvencí. Často dochází i k úplné shodě, kdy se na několika znacích všechny porovnávané sekvence shodují, ale pokud tomu tak není, je vybrán jednoduše takový znak, který snižuje hodnocení co nejméně.

Aby to nebylo až tak jednoduché, je zapotřebí počítat s tím, že pokud vybereme znak, který je akceptovatelný u většiny sekvencí, pak u těch, kde akceptovatelný není je nutné buď přidat mezeru, která zastupuje chybějící znak. Nebo pokud to situace neumožňuje (každé přidání mezery natahuje sekvenci a může způsobit problémy při zarovnání na celém zbytku sekvence) ponechat nevyhovující znak, což prakticky znamená vychýlení se od ideálního stavu. Takovým výchylkám se snažíme vyvarovat, ale ne vždy je to úplně možné. Proto při používání tohoto algoritmu se nastavují parametry podle potřeby kvality zarovnání. Zadávané parametry ovlivňují především právě ohodnocení rozdílných znaků a mění počty mezer vůči nevyhovujícím znakům.

4 Formáty sekvencí

Při zpracovávání sekvencí se setkáváme s mnoha formáty, ve kterých jsou uloženy. Nejjednodušším formátem je zřejmě jednoduchý text, kdy je celá sekvence v jednom textovém řetězci. Pokud ovšem pracujeme s konkrétním kusem DNA kódu například některé rostliny, pak musíme odlišovat například, kterého chromozomu se sekvence týká, v jakém je rozsahu, detaily o původu, vzniku dat. A těchto informací může být mnohem více, což předznamenává vznik standardizovaných formátů, do kterých se sekvence přepisují.

4.1 Jednoduchý text

Nejjednodušší a nejméně častou formou přenosu genetické informace pomocí výpočetní techniky je forma prostého textu. Oficiálně není podporovaná žádnou mezinárodní databází. Tato forma se používá spíše pro uchovávání dočasných, nějakým způsobem zajímavých dat, která jsou dále zpracovávána a tento formát slouží pouze jako mezičlánek. Příkladem takového formátu je ukazuje obrázek 13.

```
CTTAATAACTAATACTATAACATTGGGGCTGGTGAGATGGCTCAGTGGGT
AAGAGCACCCGACTGCTCTTCCGAAGGTCCAGAGTTCAAATCCCAGCAAC
CACATGGTGGCTCACAACCATCCGCAACATTTTTTTACTGCCCCCCCCC
```

Obrázek 13: Textový formát DNA sekvence

4.2 Fasta

Ve formátu *fasta* je uveden nejdříve samostatný řádek textu, počínající znakem „větší než“(‘>’). Celý první řádek označuje sekvenci, její původ, význam a identifikátor. Jednotlivé bloky sekvencí se rozpoznají ve velkém bloku právě podle onoho identifikačního řádku. Kvůli tomuto rozdělení začíná vždy právě znakem „větší než“. Dále se doporučuje dodržet maximální délku 80 znaků na jeden řádek. Příklad takového formátu je vidět na obrázku 14.

```
>gi|532319|pir|TVFV2E|TVFV2E envelope protein
ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
QIWQKHRTSNDSALILLNKHYNLTVTCKRPGNKTVLPVTIMAGLVFHSQKYNLRLRQAWC
```

Obrázek 14: Sekvence DNA ve formátu Fasta.

4.3 EMBL

Tento formát jde v identifikaci sekvence ještě o kus dále. Pro každý záznam v databázi existuje totiž značné množství informací určující konkrétní vlastnosti sekvence. Z těch významných je to jednak specifikace v databázi, dále specifikace organismu, hlavička sekvence, skupina zabývající se jejím zkoumáním a podobně. Mimo jiné se v hlavičce záznamu objevují i komentáře a dodatky, které se sekvence týkají. Po uvedení informací v hlavičce následuje samotný kód sekvence, který je typicky dělen na bloky po 10 znacích oddělených mezerami. Jako stručný příklad, jak vypadá takovýto záznam slouží obrázek 15.

```
ID   X56734; SV 1; linear; mRNA; STD; PLN; 1859 BP.
AC   X56734; S46826;
DT   12-SEP-1991 (Rel. 29, Created)
DT   25-NOV-2005 (Rel. 85, Last updated, Version 11)
DE   Trifolium repens mRNA for non-cyanogenic beta-glucosidase
SQ   Sequence 1859 BP; 609 A; 314 C; 355 G; 581 T; 0 other;
aacaaacca aatatggatt ttattgtagc catatttgct ctgtttgtta ttagctcatt 60
cacaattact tccacaaatg cagttgaagc ttctactctt cttgacatag gtaacctgag 120
tcggagcagt tttcctcgtg gcttcatctt tggtgctgga tcttcagcat cccaatttga 180
aggtgcagta aacgaaggcg gtagaggacc aagtatttgg gataccttca cccataaata 240
```

Obrázek 15: Sekvence ve formátu EMBL

4.4 GENBANK

Asi nejlépe z výše uvedených formátů je vybaven identifikací právě formát GENBANK. Jak bylo uvedeno u formátu EMBL, také zde je v hlavičce záznamu spousta informací, které určují vlastnosti uložené DNA sekvence. Odlišnost od EMBL je především v definicích jednotlivých vlastností. A dále

```
LOCUS       SCU49845     5028 bp    DNA             PLN       21-
JUN-1999
DEFINITION  Saccharomyces cerevisiae TCP1-beta gene, partial cds,
and Axl2p
            (AXL2) and Rev7p (REV7) genes, complete cds.
ACCESSION   U49845
VERSION     U49845.1  GI:1293613
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
ORIGIN
1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac ggaaccattg
61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct
121 ctgcatctga agccgctgaa gttctactaa gggtggataa catcatccgt
gcaagaccaa
```

Obrázek 16: Sekvence DNA ve formátu GENBANK

je sekvence opět formátována po 10 znacích oddělených mezerami. Pouze čísla označující první znak na řádku jsou vlevo, tedy na opačné straně, než tomu bylo u formátu EMBL. Zkrácený příklad záznamu ve formátu GENBANK můžete vidět na obrázku 16.

4.5 Shrnutí

Bylo uvedeno několik způsobů uložení DNA sekvencí. Každý ze způsobů s sebou nese specifickou strukturu záznamu. Tato struktura je pro daný formát předem daná a neměnná. Proto při zpracovávání dat programem musíme myslet na podporu různých typů záznamů. Je důležité, aby program dokázal zpracovat mezinárodně dodržované formáty, neboť pokud bude uživatel nucen nejprve změnit formát svého DNA záznamu na podporovaný a následně výsledek opět formátovat zpět do původního, pak je jisté, že bude spíše hledat jiný druh programu, který dokáže zpracovat právě jeho formát. Toto je tedy také jedna z důležitých vlastností programu, musí být co nejvíce flexibilní vůči zdrojovému textu.

5 OpenCL

OpenCL (otevřený počítačový jazyk) je licencován jako bezplatný standard pro využití paralelního programování především na výpočetních a grafických procesorech, ale i mnoha jiných druzích procesorů. Navazuje tak na předešlý vývojový krok, který učinila společnost Nvidia (výrobce především grafických procesorů a grafických karet). Ta vyvinula speciální knihovny pro využití paralelismu na grafických kartách. Toto spojení knihoven nese název *CUDA* (v originále: compute unified device architecture).

CUDA

Cílem bylo umožnit využití výpočetního potenciálu, jaký grafické karty mají pokud možno na maximum. Výhodou silnějších (dražších) grafických karet, které mají podporu právě pro CUDA architekturu je mimo jiné jejich počet jader, určených pro výpočet. Například grafická karta „Quadro FX 5800“ má těchto jader přesně 240. Toto číslo uvádí, kolik výpočtů může probíhat současně v jednom čase na různých místech v rámci této grafické karty. Jelikož se výrobce specializuje také na zpracování obrazu, je téměř jasné, že této možnosti paralelního výpočtu se bude využívat většinou právě při práci s obrazem. A to tak, že se rozdělí výpočet jednotlivých bodů obrazovky na jednotlivá jádra, abychom maximálně urychlili výpočet. Tedy pokud jako příklad použijeme monitor s rozlišením 800 na 600 bodů, máme celkem 48 000 bodů, které nám tvoří obraz. Při práci na jednom procesoru bychom museli postupně nastavit vlastnosti každého bodu, tedy pravděpodobně bychom v cyklu provedli nějaký výpočet a následně přiřazení hodnoty. Dohromady asi 48 000 cyklů. V případě použití všech CUDA jader, které výše zmiňovaná karta nabízí, můžeme každému jádru přenechat 48 000/240 cyklů. Což ve výsledku je 2000 cyklů na jádro. V tomto ideálním případě by výpočet trval jen jednu dvěstěčtyřicetinu původního času. Tento příklad je sice idealizovaný, ale prakticky vysvětluje hlavní myšlenku, která podnítila vznik této technologie.

OpenCL

Jako následník architektury CUDA si s sebou nese podobné principy, které jsou v některých směrech více propracované. Snaží se vývojářům software umožnit přenositelný a snadný přístup k síle různých druhů procesorů. *OpenCL* podporuje širokou škálu aplikací. Snaží se o nízko úrovňový přístup, díky němuž dosahuje vysokých výpočetních výkonů. Sestává z aplikačního rozhranní, které koordinuje paralelní programování napříč různými druhy procesorů. *OpenCL* standardy jsou:

1. Podporuje jak datově, tak i úlohově založené paralelní programování
2. Využívá podskupinu normy ISO C99 s rozšířením pro paralelní programování
3. Numerické výpočty jsou založeny na normě IEEE 754 (pro dvojkovou aritmetiku v plovoucí řádové čárce)

4. Definuje konfigurační profily pro vestavěná i přenosná zařízení.
5. Účinně spolupracuje s *OpenGL*, *OpenGL* ES a ostatními grafickými API.

Specifikace je rozdělena na specifikaci jádra, která se využívá při implementaci. Dále na specifikaci systému, pro který bude *OpenCL* využit, tedy buď přenosné, nebo vestavěné systémy. A na specifikaci týkající se druhu použitého jádra, s čímž jsou spojena konkrétní rozšíření pro použitou architekturu.

5.1 Architektura

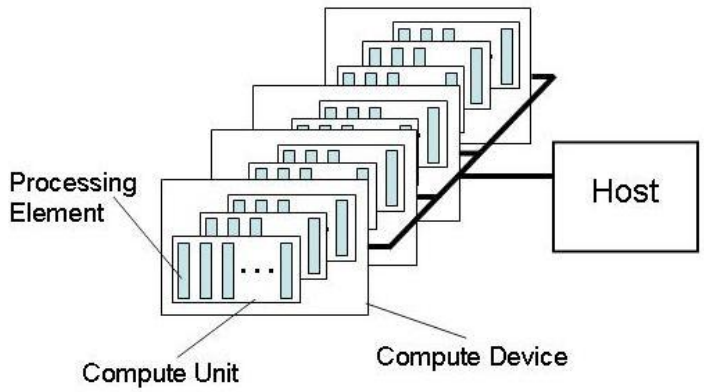
OpenCL je v podstatě *framework* pro paralelní programování, který zahrnuje jazyk, aplikační rozhranní, knihovny a běžící systém (*runtime system*) pro podporu softwarového vývoje. Například programátor může napsat program určený pro grafické procesory, aniž by musel převádět své algoritmy do 3D aplikačního rozhranní jako například pro *OpenGL* nebo *DirectX*. Cílem *OpenCL* je, aby mohli programátoři psát přenositelný a velice efektivní kód. Pro popsání jádra, na kterém architektura *OpenCL* stojí, se využívají čtyři modely:

1. Platformní model
2. Vykonávací model
3. Paměťový model
4. Programový model

Tyto modely si zde jen zběžně představíme, abychom měli o jádru *OpenCL* konkrétnější představu.

5.1.1 Platformní model

Tento model sestává z hostitelského počítače připojeného k jednomu nebo k více *OpenCL* zařízením. *OpenCL* zařízení se dělí na jednu nebo více výpočetních jednotek, které se dále dělí na jednu nebo

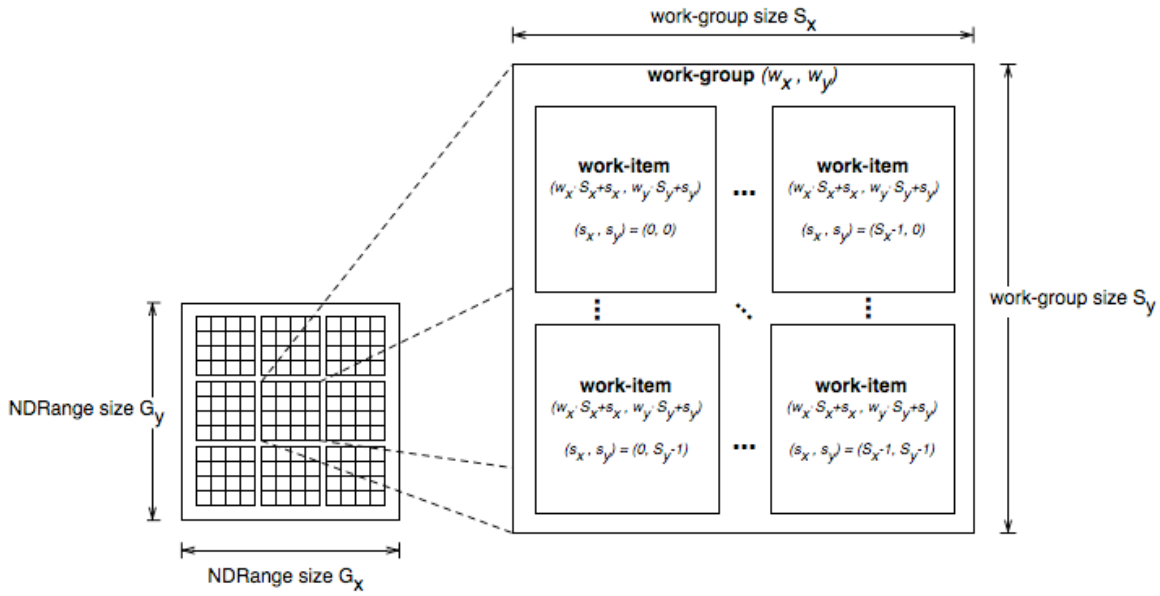


Obrázek 17: Model platformy: Jeden host, s jedním nebo více výpočetními zařízeními. Každé s jednou nebo více výpočetními jednotkami, každá s jedním nebo více procesními elementy

více procesních částí (pozn. v originále „processing elements“). Aplikace, která na takto připojeném hostitelském počítači běží, odesílá příkazy z hostitele na procesní části, kde dochází k provedení výpočtu. Procesní části bez výpočetní jednotky provádí samostatný proud instrukcí jako SIMD jednotky (provádí blokovaný krok s proudem instrukcí) nebo jako SPMD jednotky (každá jednotka podporuje její vlastní programový čítač).

5.1.2 Vykonávácí model

Provádění *OpenCL* programu probíhá ve dvou částech: první částí je jaderná, prováděná na jedné nebo více *OpenCL* jednotkách a druhou částí je hostitelský program prováděný na hostovi. Hostitelský program definuje kontext pro jádro a vede jeho provádění. Hlavní věcí modelu vykonávání v *OpenCL* je definování toho, jak funguje provádění instrukcí na jádru. Když je jádro požádáno o provedení na hostovi, definuje se indexovaný prostor. Pro každý bod v tomto indexovaném prostoru se využívá instance jádra. Tato instance se nazývá pracovní jednotka (pozn. v originále tzv. „*work-item*“) a je identifikována pomocí polohy onoho konkrétního bodu v indexovaném prostoru, ten tedy poskytuje globální identifikátor pro pracovní jednotky.



Obrázek 18: Ukázka indexovaného prostoru s instancemi pracovních jednotek

5.1.3 Paměťový model

Výše zmíněné pracovní jednotky mají možnost přistupovat k čtyřem různým druhům paměti. Prvním z nich je globální paměť, do které mají možnost zapisovat/číst veškeré pracovní jednotky. Dalším druhem paměti je paměť konstantní. Jedná se o paměť alokovanou hostem po dobu běhu jádra, tato paměť je přístupná pouze konkrétnímu hostovi, který si ji alokoval. Třetím druhem paměti je paměť lokální. Slouží pro celou pracovní skupinu (tvořená skupinou pracovních jednotek), tedy sdílejí se zde data, která mohou využívat všichni členové skupiny. A posledním druhem paměti je paměť privátní. Tato paměť patří k jedné konkrétní pracovní jednotce, která zde má uloženy hodnoty proměnných a tato oblast paměti je viditelná pouze pro jednotku, která ji alokovala, ostatním je skryta a nemohou k ní tedy přistupovat.

5.1.4 Programový model

Programový model používá paralelního programování, kdy jsou dvě možnosti, jak k němu přistupuje. První z nich je explicitní definování ve zdrojovém kódu, kde programátor specifikuje počet pracovních jednotek určených k paralelnímu zpracování. Současně musí i definovat rozložení těchto jednotek do pracovních skupin. Druhou, pro programátora pohodlnější možností je nechat vše definovat implicitně. Jediné co musí definovat je počet jednotek určených k paralelnímu zpracování, rozdělení do skupin už řeší implicitně *OpenCL*.

Další významnou částí programovacího modelu je synchronizace mezi pracovními jednotkami v rámci jedné pracovní skupiny. Je řešena pomocí tzv. bariér, kdy je stanovena pro danou skupinu bariéra, něco jako záchytný bod, kterého musí dosáhnout všechny jednotky, aby mohli pokračovat dále v práci. Tedy dalo by se jednoduše říci, že zde jednotky srovnají krok v rámci celé své skupiny. Synchronizace se používá vždy pouze v rámci pracovní skupiny, mezi jednotlivými pracovními skupinami není žádný takovýto mechanizmus synchronizace zapotřebí.

6 Vlastní implementace

Základem zarovnávacího programu je algoritmus, který bude tvořit hlavní výpočetní jednotku, která bude zajišťovat rychlost zarovnávání. Na výběr bylo široké spektrum algoritmů, ty nejvýznamnější byly uvedeny v předchozích kapitolách. Po testech na dobu provádění se dostal do čela algoritmus *Clustal*.

Clustal je implementován v mnoha již hotových projektech, které se aktuálně k zarovnávání používají, jedná se především o komerční nástroje, tedy takové, na které byly vynaloženy nemalé finanční prostředky a pracovaly na nich celé týmy programátorů. Mimo těchto komerčních jsou k vidění i ke stažení také projekty pod licencí GPL, která zajišťuje dostupnost programu i s otevřenými zdrojovými kódy. Tyto programy bylo nutné vyzkoušet a otestovat jak jsou na tom jak s rychlostí, tak také s kvalitou zarovnávání. Jednou z nejlepších variant se ukázal projekt *Clustal* z Irské univerzity dostupné z internetových stránek, kde lze celý program se zdrojovými kódy stáhnout (zdroj [7]).

6.1 Program Clustal

Autory tohoto programu jsou členové irské univerzity (oficiání kontakt na adrese: http://bioinf.ucd.ie/index.php). Program je vyvíjen přibližně již od roku 1988. Spolupracovali na něm dvě desítky programátorů, především se jednalo o studenty doktorského studijního programu. V poslední verzi, kterou lze nalézt volně ke stažení na internetu viz. zdroj [7] se nacházejí jak již zkompilované soubory, tak i zdrojové kódy, ze kterých byla kompilace provedena.

Po stažení zdrojových souborů narazíme na odlišnosti v názvosloví. Jeden typ programu má název „*ClustalW*“ a druhý „*ClustalX*“. Jedná se o rozdíl především v grafické podobě spustitelného programu. Ta verze končící písmenem „W“ je konzolová aplikace, tedy program řídíme zadáváním příkazů do konzole, kdežto verze končící písmenem „X“ má grafický výstup a tedy vše probíhá na reakci jak myši tak také klávesnice. Původní program bylo možné spustit pouze jako konzolovou aplikaci. Grafickou podobu autoři přidali již někdy kolem roku 1997, ale teprve před nedávnem bylo použito pro grafické rozhraní frameworku *Qt* od nokie a od té doby vypadá aplikace více než moderně. Na vylepšení programu autoři pracují čistě na konzolové verzi, což jsem se dozvěděl od jednoho ze spoluautorů (kontaktní emailová adresa: andreas.wilm@ucd.ie). Důvod k takovému způsobu práce je, že kompilace konzolové aplikace trvá řádově minutu až dvě, ale kompilace grafické aplikace se protahuje již řádově na pět až osm minut a tedy velmi zdržuje při vývoji.

Samotné jádro programu je společné jak pro verzi „W“ tak i pro verzi „X“ a skládá se z několika částí. Základem je hlavní soubor *main.cpp* který se stará o běh celé aplikace. Nejdůležitější součástí je pak soubor *Clustal.cpp*, který už vytváří podle potřeb objekty pro dané akce, které si uživatel zvolil ať už použitím konzole nebo grafické aplikace. Jednotlivé objekty, které je možné

vytvořit, se starají od párového zarovnávání přes zarovnávání mnohonásobné až po práci se soubory. Veškeré zdrojové kódy jsou umístěny do složek tak, aby bylo možné identifikovat jejich funkčnost, tedy nikoho pak nepřekvapí, že ve složce „*pairwise*“ jsou zdrojové kódy starající se o párové zarovnávání a takto je zbytečné pokračovat ve výčtu jednotlivých zdrojových kódů. Důležité je, že program kopíruje algoritmus *Clustal*, který byl uveden v kapitole 3.2.2 a je tedy přesně tím, co jsme od implementace tohoto algoritmu očekávali.

Hlavní nevýhodou jak samotného programu, tak i projektu jako celku je dokumentace. Bohužel autoři dokumentaci značně podcenili v ranné fázi vývoje a tento nešvar doposud nikdo z nich nenapravil. Což je až trestuhodné, pokud bereme v potaz, že se jedná o volně šiřitelný projekt, který využívá několik výzkumných týmů po celém světě a očekává se tedy, že si jej jednotlivé týmy budou modifikovat pro vlastní potřeby výzkumu. Díky tomuto nedostatku je zapotřebí mnohem většího úsilí a více času na jakékoliv změny než by tomu bylo s dokumentací.

6.2 Návrh vylepšení

Program je sám o sobě programem jedním z nejrychlejších z volně dostupných určených pro zarovnávání DNA kódu. Ale samozřejmě tak jako ve sportu i v programování lze posunovat dosaženou laťku o kousek dál. Dalším urychlení by bylo možné dosáhnout použitím moderních technik jakými je paralelní programování, které se dostává do popředí především v programech, které se zabývají grafikou. Pro takovéto programy byly navrženy speciální knihovny (přecházejí postupně až na hranici *frameworků*), mezi ty moderní patří *OpenCL* (kapitola 5), které se nabízí k použití a vyzkoušení nakolik bude pomocí tohoto principu program urychlen.

Pro dosažení co největšího urychlení aplikace není nutné přepisovat celý program do podoby, jakou předepisuje *OpenCL*, ale postačuje analyzovat, kde dochází k největšímu vytížení a kde by tedy mohlo jeho použití co nejvíce zapůsobit na rychlost chodu programu. Tyto místa jsou předem daná algoritmem samotným. Jedná se o velkou náročnost při párovém zarovnávání, kdy jsou v cyklu procházeny prakticky téměř všechny kombinace obou sekvencí. Na tomto místě tedy dochází k maximalizaci využití jak paměťových prostředků, tak i výpočetního času procesoru. Pro urychlení je vhodné paralelizovat cyklus na několik nezávislých jednotek, které cyklus provedou ideálně v podstatně kratším čase. Kromě párového zarovnávání je zde zlomovým bodem pro náročnost na výpočet zarovnání mnohonásobné, kde dochází k ještě k většímu nárůstu potřeby paměti a procesorového času, zde by měl paralelismus také sehrát významnou roli pro dobu a náročnost výpočtu.

Kromě výše zmíněných kritických bodů je možné aplikaci vylepšovat dále prozkoumáním, kde ještě dochází k nárůstu potřeb na výpočetní zdroje a následnou implementací daného kódu pomocí OpenCL. Optimalizací takovýchto míst, ale neočekáváme velký rozdíl v době výpočtu. Proto vylepšení těchto částí bude závislé na čase, který při programování zůstane.

6.3 Popis implementace

Samotná implementace není z programátorského hlediska úplně triviální. Nejedná se pouze o napsání kódu v C/C++ který zkompilujeme a vytvoříme takto spustitelný program. Jelikož je nezbytné použití knihoven *OpenCL*, musíme náš kód i pracovní prostření přizpůsobit pro práci s nimi.

6.3.1 Pracovní prostředí

Prostředí, ve kterém lze vyvíjet s použitím knihoven *OpenCL* je předem dáno jednak operačním systémem a dále podporovanými nástroji. Jako implementační prostředí jsem zvolil operační systém Microsoft Windows, kvůli jeho rozšiřitelnosti a především dostupnosti nástrojů od společností ATI a NVIDIA. Pro zpřístupnění knihoven je nutné nejprve získat vývojové prostředí od výrobce hardware, na kterém chceme vývoj provádět. V mém případě se jedná o výrobce ATI (AMD), který poskytuje vývojové prostředí ke stažení zdarma (zdroj [8]) společně se souborem příkladů pro ukázku, jak lze vůbec s využitím knihoven programovat. Jako vývojový nástroj ATI doporučuje primárně Microsoft Visual Studio 2008 a vyšší, ale také uvádí, že postačuje překladač *g++*, kde bych byl skeptický a po vlastní zkušenosti bych také doporučil použít nástroj Microsoft Visual Studio 2008. Jak bylo uvedeno v kapitole 3.2.2 *OpenCL* dokáže využít paralelismu i na centrální procesorové jednotce. Ne jako jeho předchůdce CUDA, který byl úzce specializován pro práci na grafických procesorech. Díky této vlastnosti jsem mohl vyvíjet i přes tu skutečnost, že nevlastním grafickou kartu, která podporuje práci s knihovnami *CUDA* nebo *OpenCL*. Soupis grafických karet, které jsou podporovány lze vidět na adrese zdroje [8].

Použití pracovního nástroje Microsoft Visual Studio2008

Po stažení a nainstalování prostředí pro práci s *OpenCL* lze nalézt složku, v níž jsou umístěny příklady dodané spolu s prostředím. Tyto příklady doporučuji otevřít přes nabídku *Soubor* a *otevřít projek*, kde jako cestu určíme cestu k souboru, který obsahuje všechny ukázkové příklady (povětšinou ve složce *„samples“* soubor *„OpenCLSamples.sln“*). Po úspěšném načtení projektu máme k dispozici celkem přibližně 32příkladů, ve kterých nalezneme povětšinou základní algoritmy, implementované pomocí knihovny OpenCL. Pokud chceme tvořit vlastní zdrojový kód, který bude využívat této knihovny, pak se doporučuje vytvořit si nový projekt v rámci daného *„solution“* (*OpenCLSamples.sln*), neboť zde je přístupná knihovna <CL/cl.h>. Jinde ji není možné použít (vycházím z vlastního vývojového prostředí, které se může lišit od ostatních, díky tomu, že nevlastním oficiálně podporovanou grafickou kartu) aniž by bylo nutné přibalovat ke zkompilovanému programu také již hotové *„dll“* knihovny potřebné k běhu. Po vytvoření nového projektu je možné kombinovat jednak principy uvedené v ukázkových příkladech a také vše, co prostředí *OpenCL* dovoluje. Přehled metod a veškerých možností nalezneme viz. zdroj [9].

Protože mi pracovní prostředí nedovoluje použít knihovnu <CL/cl.h> na jiném místě než uvnitř předem daného *„solution“*, pak bylo nezbytné vytvořit v rámci stejného „*solution*“ také samostatný projekt pro Clustal a jelikož jeho grafická verze je spojena s nástrojem Qt a kolize mezi Qt, Visual Studiem 2008 a *OpenCL* knihovnami mi nedovolila tuto verzi zkompilovat, pak jsem byl nucen se omezit pouze na verzi s konzolovým rozhraním, tedy na *ClustalW*. V rámci projektu byly vytvořeny filtry tak, aby kopírovaly strukturu, kterou navrhli samotní autoři programu, po otevření tohoto projektu ve Visual Studiu 2008 tedy vše vypadá velmi podobně jako samotné umístění programu na disku. Jediná změna je v rozdělení souborů na soubory hlavičkové a zdrojové, ostatním změnám jsem se především kvůli přehlednosti vyhnul.

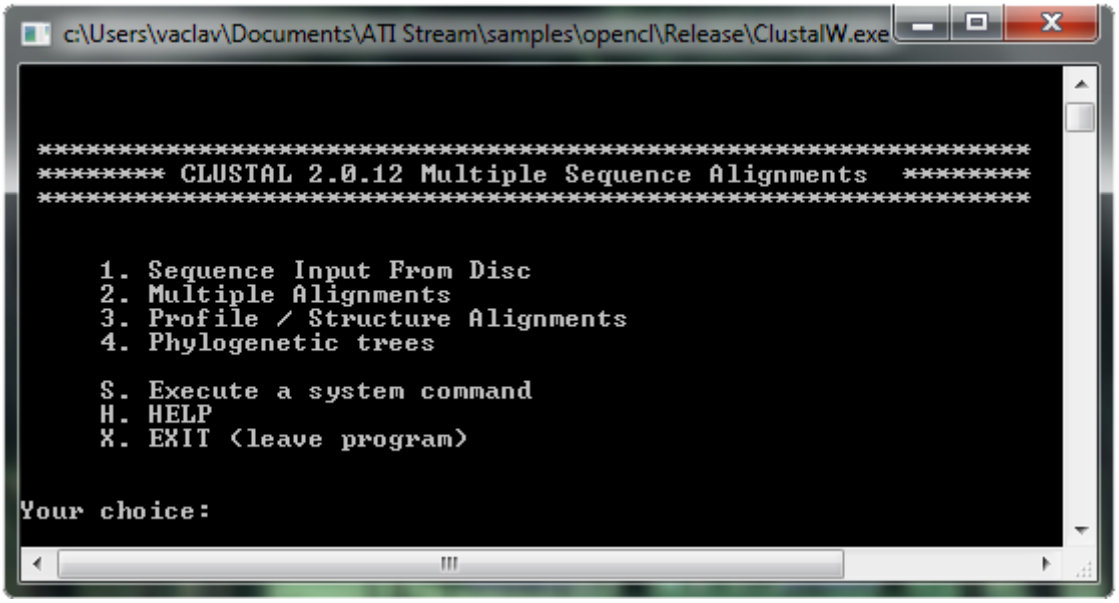
OpenCL a kernel

V jazyce C/C++ jsou používány dva druhy souborů, a to hlavičkové (h/hpp) a zdrojové (c/cpp). Pokud při programování používáme knihoven *OpenCL* narazíme na jednu zvláštnost. Tou zvláštností jsou takzvané kernely neboli jádra. Tyto jádra si můžeme představit tak, jako v C/C++ funkce. Tedy pokud napíšeme zdrojový kód pro jádro, pak je to podobné jako bychom napsali zdrojový kód funkce. Pro definici jádra nejprve použijeme klíčové slovo „*__kernel*“ následuje návratová hodnota a po ní, tak jak je zvykem v C/C++ název jádra spolu s definicí parametrů. Deklarace jádrové funkce může tedy vypadat následovně: „*__kernel void priklad(__global uint parametr)*“. Pokud jádru přidáme i nějaké tělo, aby vykonávalo alespoň jednoduchou operaci, pak je třeba jej pomocí *OpenCL* funkce uvést do provozu. Nelze tedy tak jako u volání funkce zavolat jádro. Nejprve je nutné z jádra utvořit spustitelný kód. Dále přiřadit hodnoty všem parametrům, které jádro má. Pokud je jader více, seřadíme si je do front. A pokud máme spustitelné jádro i nastavené parametry, je možné vyvolat vykonání našeho jádra. Po vykonání převedeme výsledky do lokálních/globálních proměnných a můžeme dále pracovat v normálním C/C++ režimu.

Pro jádra je vhodné vytvořit soubory s koncovkou „*cl*“, tedy soubor bude vypadat takto: „*nase_jadro.cl*“. V těchto souborech můžeme mít i více jaderných funkcí najednou. Druhým způsobem je kód jádra zapsat jako řetězec do proměnné, z které za pomocí funkcí *OpenCL* vytvoříme spustitelné jádro stejně tak, jako by bylo zapsané do souboru. Tato metoda je velice vhodná, pokud se jedná o krátké jádro, kdy je přehlednější ponechat jeho tělo přímo ve zdrojovém kódu C/C++.

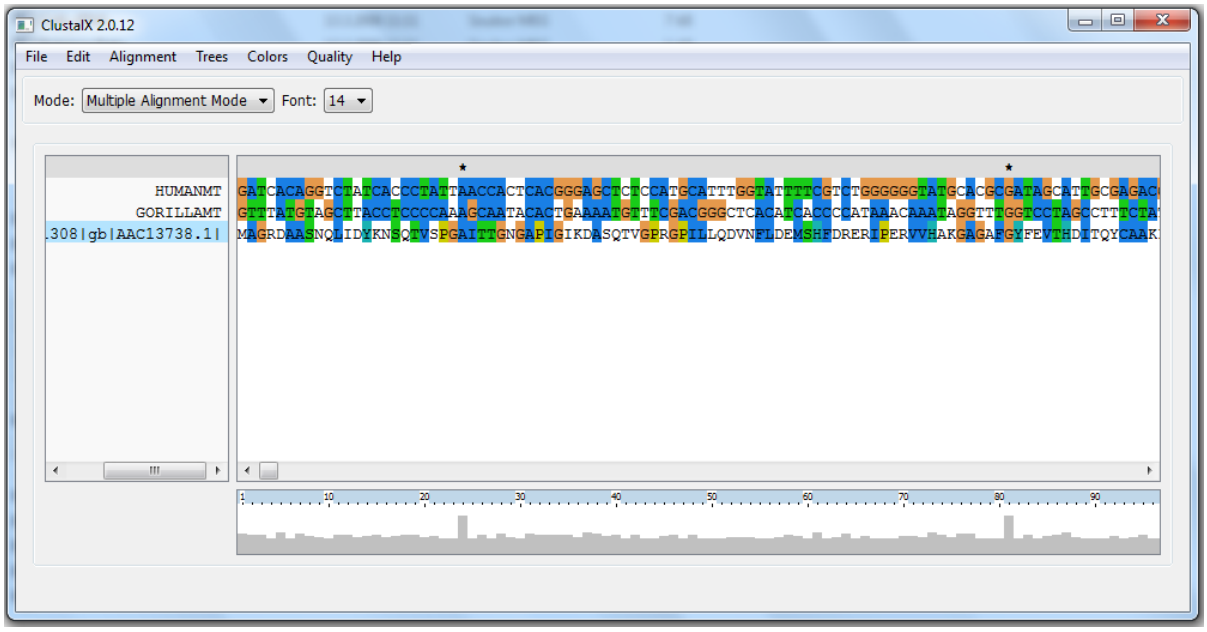
6.3.2 Výsledný program

Program, který je výsledkem zkompilování upraveného zdrojového kódu má bohužel pouze jednoduché uživatelské rozhranní, které nemá sice vliv na provádění jednotlivých funkcí programu, ale nepůsobí tak moderně jako verze grafická. Přesto po spuštění vypadá program stejně jako na obrázku 19.



Obrázek 19: Spuštěný program ClustalW

Spolu s tímto upraveným programem jsou k dispozici na disku přiloženém k této bakalářské práci také programy původní. Zde lze tedy nalézt i verzi s grafickým uživatelským rozhranním, kterou lze vidět na obrázku 20.



Obrázek 20: spuštěný program ClustalX s importovanými sekvencemi

Tedy po spuštění upraveného programu, který je vidět na obrázku 19 lze vybrat z nabídky různých akcí (podrobněji se všemi možnostmi seznamuje zdroj [6]). Ta nejdůležitější je provedení zarovnání zadaných sekvencí. Abychom zarovnání mohli provést, je nutné mít speciálně upravený soubor se zdrojem sekvencí. Program vyžaduje mít sekvence pohromadě v jednom souboru. Tedy pokud máme například zdrojový soubor ve formátu *fasta*, pak stačí za poslední řádek jedné sekvence

nakopírovat sekvenci další a takto postupně pro všechny provést jejich sloučení do jednoho souboru. Program rozpozná sekvence podle značek, kterými vždy začínají. V případě, že by nebyl dodržen správný formát (pokud například provedeme chybu při slučování sekvencí), pak program vypíše, k jaké chybě došlo. Dokáže rozpoznat i na které konkrétní sekvenci nastala chyba. Uživatel je tedy velmi dobře informován o důvodech selhání akce. Pokud se akce podaří, je vypsána informace o názvech a parametrech sekvencí, které se podařilo načíst. Po načtení následuje návrat do hlavního menu.

Po úspěšném načtení sekvencí a návratu do hlavního menu vybereme akci, kterou chceme provést. 2. položka v menu provede zarovnání načtených sekvencí. Před samotným provedením zarovnání je uživatel dotázán na jméno souboru, kam se má výsledek uložit. Soubor je opatřen koncovkou „aln“ což je anglická zkratka pro slovo zarovnaný. V takto vygenerovaném souboru je pak možné sledovat, k jakým změnám mezi sekvencemi došlo. Soubor je možné otevřít v jakémkoliv textovém editoru, obsahuje pouze textové informace (podobně jako zdrojové soubory, které uživatel zadává k provedení zarovnání).

6.4 Porovnání doby výpočtu

Pro zjištění, zda bylo vylepšení programu úspěšné, bylo nutné provést několik testů, které by prokázaly, jak se liší upravený program od programu původního a také jak si stojí proti konkurenci. Testování jsem rozdělil tak, aby bylo možné sledovat rozdílné chování programu pro krátkou a pro dlouhou sadu sekvencí. Neboť při zarovnávání krátkých sekvencí nedochází k velkým nárokům na paměť. Kromě těchto základních testů, které měly za úkol ukázat, jak se programy chovají při zarovnávání malého počtu sekvencí, které se liší délkou znaků, jsem provedl ještě jiné testování. Jednalo se o test chování všech programů při zarovnávání většího počtu sekvencí vůči sobě. Tyto testy měly za úkol ověřit, jak se bude chování programů lišit od testu jen s několika sekvencemi. Pro testování jsem zvolil jako pracovní prostředí operační systém Microsoft Windows. A testoval jsem jak na svém notebooku, tak také na školním počítači, který měl nainstalovanou grafickou kartu, která přímo podporuje technologii *OpenCL*.A ještě jsem použil pro testování volně dostupný počítač v centru výpočetní techniky, které se nachází na naší fakultě. V následující tabulce jsou uvedeny parametry testovacích počítačů.

Název stanice	**Grafická karta**	**Procesor**	**Paměť RAM**	**Operační systém**
PC1	ATI HD 2600	AMD X2 2.00GHz	DDR II 3GB	Windows 7 32bit
PC2	Nvidia GeForce 7300 GT	Intel E6750 2.66GHz	DDR II 2GB	Windows XP 32bit
PC3	Nvidia GeForce GTX 285	Intel E8600 3.33GHz	DDR II 4GB	Windows 7 64bit

Zarovnávání dlouhých sekvencí

Pro testování jsem vybral 3sekvence, kdy se jedná o část DNA člověka, gorily a myši domácí. Sekvence byly dlouhé v rozmezí 16299 – 16569 znaků. Jako pracovní stanice pro testování byly použity uvedené počítačové sestavy z předchozí tabulky.

Pracovní stanice	**ClustalX originál**	**ClustalW originál**	**ClustalW upravený (GCC)**	**ClustalW upravený (VS 2008)**	**Mega 4.1**
PC1	471s	452s	315s	1 123s	778s
PC2	185s	181s	120s	421s	xxx
PC3	149s	142s	91s	334s	217s

Zarovnávání středně dlouhých sekvencí

Pro zarovnávání středně dlouhých sekvencí jsem zvolil recyklovat sekvence z prvního testu, kdy jsem pouze omezil počet znaků u každé sekvence. V tomto testu tedy bylo použito 6000 prvních znaků z části lidské DNA, 6000 prvních znaků z části DNA gorily a také 6000 prvních znaků z části DNA myši domácí. K recyklaci jsem se rozhodl pro zachování co největší kompatibility výpočtu, aby se výsledky neodlišovaly kvůli jinému rozložení znaků.

Pracovní stanice	**ClustalX originál**	**ClustalW originál**	**ClustalW upravený (GCC)**	**ClustalW upravený (VS 2008)**	**Mega 4.1**
PC1	55s	53s	37s	125s	98s
PC2	26s	25s	18s	55s	xxx
PC3	21s	20s	14s	48s	29s

Zarovnávání krátkých sekvencí

Pro poslední test, který měl za účel zjistit, jak si stojí jednotlivé programy při zarovnávání opravdu krátkých sekvencí, bylo využito opět původních tří DNA kódů. Tedy opět byly použity DNA kódy člověka, gorily a myši domácí. Pro tento test bylo použito prvních 600 znaků.

Pracovní stanice	ClustalX originál	ClustalW originál	ClustalW upravený (GCC)	ClustalW upravený (VS 2008)	Mega 4.1
PC1	1s	1s	1s	5s	2s
PC2	1s	1s	1s	3s	2s
PC3	1s	1s	1s	2s	1s

Zarovnávání velkého počtu sekvencí

Pro tento test mi poskytla data Eva Chovancová M.Sc. z Masarykovy univerzity z oddělení experimentální biologie, která zaslala několik balíků dat pro testování. Balíky sestavila tak, aby bylo možné otestovat programy na datech, která se zarovnávají obtížně. Tyto balíky mají otestovat programy, jak budou zvládat větší počet sekvencí, než v prvních testech, kde se jednalo jen o tři sekvence, kde jsem měnil jejich délku. V elektronickém dopise, který obsahoval jako přílohu tyto testovací data, přiložila také krátký popis faktorů, které mají vliv na obtížnost zarovnávání:

- příbuznost sekvencí (blízce příbuzné je lehčí zarovnat)
- množství sekvencí
- délka sekvencí (pár dlouhých sekvencí může ztížit i jinak relativně jednoduchý alignment)

V prvním balíku bylo celkem 241 sekvencí, což je podle autorky malý počet. Navíc se jednalo o krátké sekvence (průměrně 300znaků), které patřily do jedné proteinové rodiny. Tedy byly si dost podobné. A proto první balík dat není extrémně náročným. Pro jeho otestování jsem již použil pouze osobní počítač, neboť testování na dalších počítačích nemění procentuální rozdíly v době výpočtu mezi jednotlivými programy.

Pro první balík dat jsou výsledky následující:

Pracovní stanice	ClustalW originál	ClustalW upravený (GCC)	Mega 4.1	Muscle
PC1	293s	215s	754s	483s

V druhém balíku jsou data podstatně obtížnější na provedení zarovnání než v předchozím příkladě. Balík obsahuje celkem 1070 sekvencí opět z jedné proteinové rodiny (úzce příbuzné). Navíc jsou některé sekvence již delšího rozsahu (kolem 650znaků) a podle specifikace uvedené u balíku je dosti stěžujícím faktorem, pokud mezi krátké sekvence zamícháme pár dlouhých. Výpočet se tím stává mnohem náročnějším.

Pro druhý balík dat jsou výsledky následující:

Pracovní stanice	ClustalW originál	ClustalW upravený (GCC)	Mega 4.1	Muscle
PC1	5 161s	4 062s	13 785s	28 611s

Další testovací balík dat vznikl sloučením dvou předchozích. Celkově se tedy jedná o 1311 sekvencí, které již nepatří do jedné proteinové rodiny a jejich délka je v rozsahu 300 až 600 znaků. Test je tedy opět o něco obtížnější, než byly testy předchozí.

Pro třetí balík dopadlo testování následovně:

Pracovní stanice	ClustalW originál	ClustalW upravený (GCC)	Mega 4.1	Muscle
PC1	7 696s	5 222s	19 412s	64 973s

Čtvrtým a poslední balík dat obsahuje již obrovské množství sekvencí, přesně 6901. Navíc se jedná o sekvence z několika různých proteinových rodin. Proto je tento test nejtěžším ze všech uvedených a má prokázat, zda budou výsledky podobné výsledkům předchozích testů, nebo bude upravený program zaostávat.

Tento balík jsem testoval nejprve programem *Mega 4.1*, kdy po 20 hodinách zarovnávání měl hotovo jen 20% z celkového počtu sekvencí. Odhadem tedy usuzuji, že by se doba celého zarovnání vyšplhala přes 100 hodin. Z časových důvodů jsem nemohl test dokončit, ale podle doposud uvedených testů by zde již bylo urychlení upraveného programu *ClustalW* na tolik markantní, že by se zkrátila doba výpočtu proti programu *Mega 4.1* i o několik dnů. Pokud bych uvažoval urychlení, jakého bylo dosaženo u posledního testu (testování třetího balíku dat), pak by se doba výpočtu ze 100 hodin, které by potřeboval program *Mega 4.1*, snížila asi na 27 hodin. A při srovnání programu *ClustalW* bez úprav s upraveným programem předpokládám urychlení asi o 8 hodin. Tyto odhady sice nejsou podloženy testy, ale s vysokou pravděpodobností lze předpokládat, že by se reálné výsledky od tohoto předpokladu příliš nelišily.

Hodnocení výsledků testování

Z provedených testů, které proběhly pro zjištění rychlosti zarovnávání jednotlivých programů, jsou poměrně různorodé výsledky. Testování odhalilo několik zajímavých vlastností, které jednotlivé verze programů mají. Co je na výsledcích až šokující, tak je vliv použitého překladače. Pokud jsem zapojil program, který byl přeložen pomocí překladače programu *Microsoft Visual Studio 2008*, pak byl výsledný program velmi pomalý, potřeboval obrovské množství času i na jednoduché zarovnání. Pokud jsem ale stejný zdrojový kód přeložil překladačem *gcc (*v kominaci s *g++)*, pak naopak dosahoval zřetelně nejlepších výsledků ze všech použitých programů. Je tedy otázkou, na kolik procent má vliv samotné použití *OpenCL* na rychlost výpočtu. Neboť podle všech provedených testů bych usuzoval, že jsem svými úpravami nemohl dosáhnout tak markantního urychlení. Spíše bych sázel na to, že autoři, poskytující oficiální verzi programu ClustalW a ClustalX použili při kompilaci zdrojových kódů jinou verzi překladače nežli já, a to právě způsobilo tak propastný rozdíl. Tak jako tak, je vynikající, že se podařilo dosáhnout tak velkého urychlení na všech testovaných příkladech.

Podle předpokladů se od sebe moc nelišily programy, které lze stáhnout z adresy uvedené ve zdroji [7]. Verze s grafickým uživatelským rozraním se vyrovnala verzi textové, kromě testu s dlouhými sekvencemi. Zde byl rozdíl přibližně 19 sekund při testu na PC1, což činí prodloužení doby výpočtu programu s grafickým rozhranním asi o 4% vůči textové verzi. Podobně dopadlo srovnání i na ostatních testovacích počítačových sestavách.

Naopak program *Mega 4.1* ukázal, že nepatří ke špičce v rychlosti zarovnávání. Jako plus má ovšem spoustu nastavení, které program *ClustalW* nemá. Přesto, pokud by bylo účelem zajistit co nejrychlejší zarovnání, pak by neměl v porovnání s programem *ClustalW,* přeloženým pomocí překladače *gcc* žádnou šanci.

6.5 Porovnání kvality zarovnání

Pro porovnání kvality zarovnání byly použity výsledky z testů provedených v předchozí kapitole (6.4). Všechny výsledky dosahovaly úplné shody. U programu Mega 4.1 bylo použito základní nastavení, tedy nebylo nutné nic měnit, pouze načíst sekvence a nechat program sekvence zarovnat. V kvalitě zarovnání tedy *ClustalW* nebo *ClustalX* nijak nezaostávají za programem *Mega 4.1*, což je jedno z velmi důležitých kritérií pro praktické využití programu. Porovnání kvality tedy dopadlo nad očekávání dobře. A navíc i upravený program *ClustalW* výsledky ukládá ve standardizovaném formátu. Je tedy možné kvůli rychlosti zarovnání použít program *ClustalW* a pro práci s výsledky a jejich grafické znázornění pak použít jakýkoliv jiný program, ať už *ClustalX* nebo *Mega 4.1* nebo úplně jiný.

7 Závěr

Po provedení všech testů již hotového programu jsem dospěl k závěru, že cíle práce se podařilo splnit. Program, který jsem vytvořil, dokáže zarovnat DNA sekvence opravdu nejrychleji v porovnání s dostupnými nástroji. A ke všemu provádí zarovnání se stejnou kvalitou jako konkurenční programy. Díky jeho rychlosti by mohl být nasazen v brzké době do praxe.

Jako hlavní nedostatky vidím nepřenositelnost vytvořeného programu mezi platformami. To je ovšem způsobeno zvolenými technikami, které jsou svázány s pracovním prostředím, které nelze mezi platformami přenášet. Do budoucna by tedy nebylo na škodu dokázat zdrojové kódy upravit tak, aby je bylo možné přeložit na všech platformách. Dalším nedostatkem je chybějící dokumentace, která brzdí vývoj a jakékoliv úpravy. Tento nedostatek by ovšem museli napravit autoři programu, nikoliv člověk, který pouze program vylepšuje a nezná všechny jeho součásti tak dobře jako samotný autor.

Pevně věřím, že tato práce měla smysl a poukázala na možnost využití nejmodernějších technologií (*OpenCL*) při tvorbě programů nejen s genetickou tématikou. Do budoucna se může stát základem pro ještě rychlejší a lepší programy pro zarovnávání. Lze využít jak zdrojové kódy, tak také optimalizovat zvolené metody a technologie tak, aby se dosáhlo ještě lepších výsledků.

Literatura

[1] Neil C. Jones and Pavel A. Pevzner: An introduction to bioinformatics algorithms, MIT Press, 2004, 435 s., ISBN: 0-262-10106-8

[2] Wikipedia.org: článek o algoritmu Smith-Watermann [online], [cit. 10.5.2010], dostupné z: < http://en.wikipedia.org/wiki/Smith_waterman >

[3] Benda Vladimír, Babůrek Ivan, Kotrba Pavel. Základy biologie 1.vydání, VŠCHT v Praze, 2006, 168 s., ISBN: 80-7080-587-0

[4] Ian Korf, Mark Yandell and Joseph Bedell:Blast, O'Reilly Media, 2003, 368 s., ISBN: 1600330312

[5] David Edwards, Jason Stajich, David Hansen: Bioinformatics: Tools and applications, 1. vydání, Springer, 2009, 451 s., ISBN: 0387927379

[6] Stephen Misener, Stephen A. Krawetz: Bioinformatics methods and protocols, 1. vydání, Humana Press, 2000, 500 s., ISBN: 0896037320

[7] Program Clustal: stránky pro stáhnutí program ClustalW a ClustalX [online], [cit. 10.5.2010], dostupné z : <http://www.clustal.org/download/current/>

[8] ATI SDK: Stránky věnované pracovnímu prostředí OpenCL [online], [cit. 10.5.2010], dostupné z : <http://developer.amd.com/GPU/ATISTREAMSDK/Pages/default.aspx>

[9] OpenCL dokumentace: Dokumentace k programovacímu prostředí OpenCl [online], [cit. 10.5.2010], dostupné z: <https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/>

Seznam příloh

Příloha 1. DVD (manuál, programy použité pro testování, zdrojové soubory, testovací soubory)